Exercise 4

Juuso Korhonen - 652377
ELEC-E8125 - Reinforcement Learning

October 6, 2022

1 Task 1

Implement Q-learning using function approximation. Also implement -greedy action selection. Test two different features for state representations:

1.1 (a) handcrafted feature vector

Training performance plot using handcrafted feature vector is in figure 1.

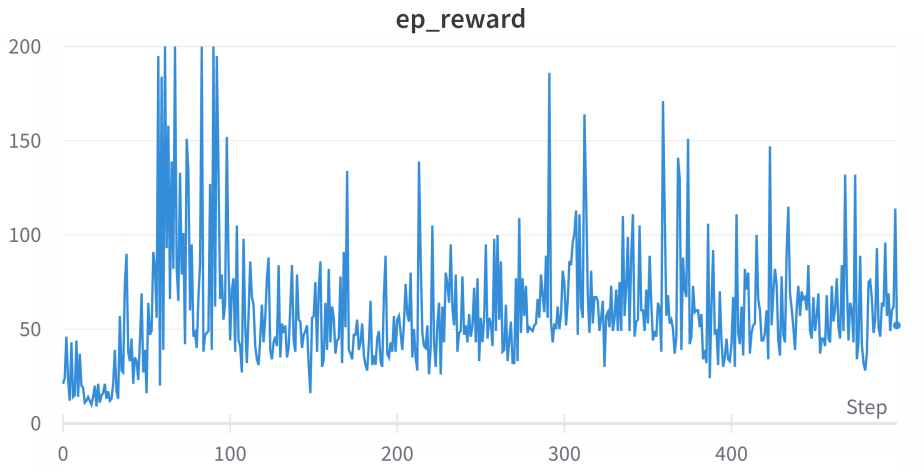

Figure 1: Train episode reward plot using hand-crafted features.

1.2 (b) radial basis function representations

Training performance plot using radial basis function representations is in figure 2.

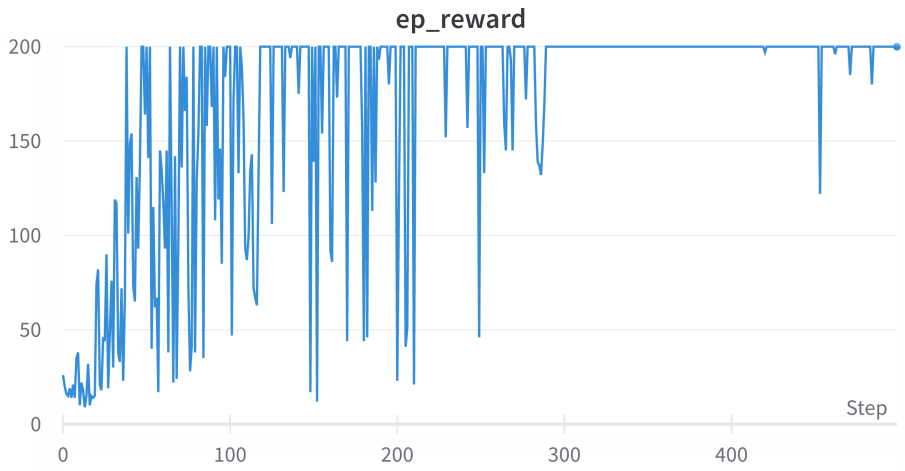

Figure 2: Train episode reward plot using using radial basis function representations.

2 Question 1.1

Q: Would it be possible to learn accurately Q-values for the Cartpole problem using linear features (by passing the state directly to a linear regressor)? Why/why not?

A: According to performance in (a) the answer is no. Learning the accurate Q-values would mean (average) reward of 200, which we can see is not happening with linear features even though they are accompanied with their absolute values.

3 Question 1.2

Q: In Task 1, we collect observed states, actions, and rewards into an experience replay buffer. During training we randomly sample mini-batches from this buffer. Why do we use the experience replay buffer, how does it affect the learning performance? Why do we sample the mini-batches randomly?

A: Experience replay buffer allows repeated use of the samples, which is required by iterative methods like stochastic gradient descent (which we are using in Task 1). Sampling randomly tries to make the samples i.i.d. (i.e. independent and identically distributed). If we would take state-action-samples in a trajectory this would obviously break the i.i.d. I.i.d is a cornerstone requirement of stochastic gradient optimization method (as local fitting would likely lead to overfitting).

4 Question 1.3

In Exercise 3, we used grid-based Q-learning to balance the Cartpole. Are grid-based methods sample-efficient compared to the RBF function approximation methods? Why/why not?

Based on the training performance plots the answer is no. With grid-based Q-learning it took approximately 5k steps to consistently start reaching the reward 200, but with RBF

function approximation method it took only about 200 steps. This is because in function approximation parameter updates based on seen states also effect unseen states, and if the function is appropriate (can approximate optimal Q-function well) this is much more sample efficient.

5 Task 2

Q: Create a 2D plot of policy (best action in terms of state) learned with RBF in terms of x and for = 0 and = 0, such that is on the horizontal axis and x is on the vertical axis. Attach the plot into your report.

A: The policy plot is shown in figure 3.

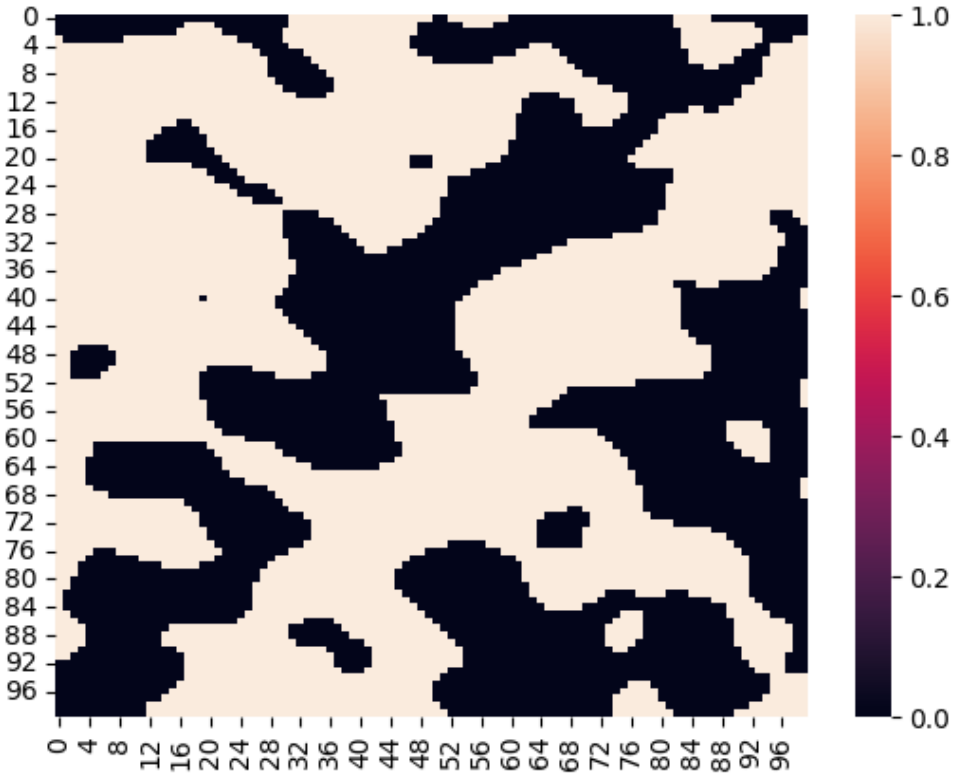

Figure 3: Plot of policy function learned with RBF.

6 Task 3

Training performance plot with DQN for cartpole environment is shown in figure 4.

Training performance plot with DQN for lunarlander environment is shown in figure 5.

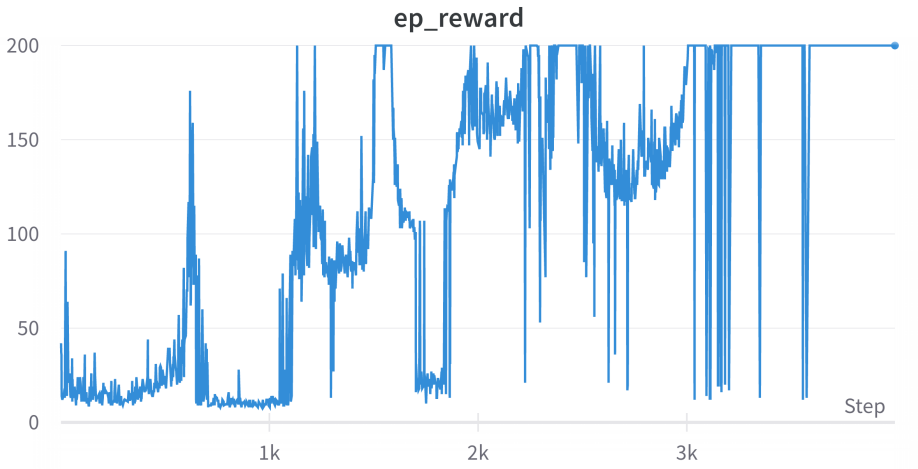

Figure 4: Train episode reward with DQN in cartpole environment.

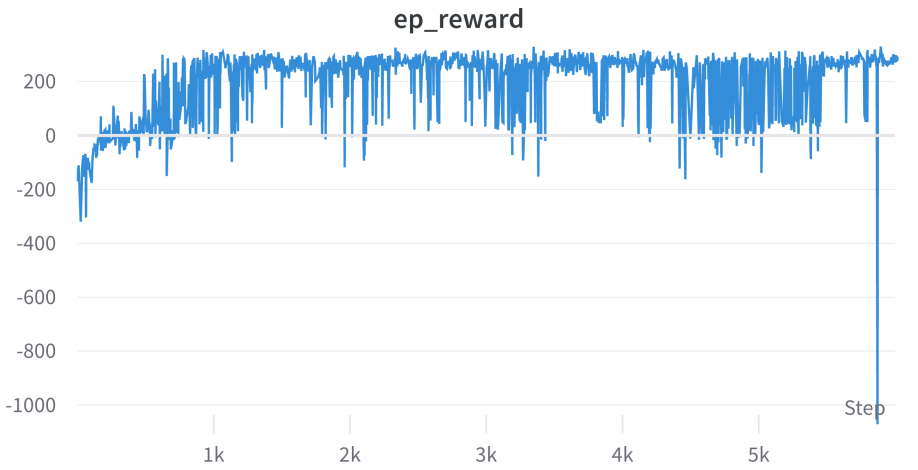

Figure 5: Train episode reward with DQN in lunarlander environment.

7 Question 3.2

The difficult step would be to find the maximum action in the next state (producing maximum value) when calculating the target Q-value. This network is not implemented to work in this way currently. It could be possible to concatenate the action to be a part of the input, so the output of the network would be the Q-value for specific state-action pair.

8 Question 3.3

Using the same network would make the target and prediction dependent, which is not what we want. If we do not stop the gradient of the target value, it would be considered as sort of a training sample (it depends on the same θ) which would break the i.i.d requirement specified in question 1.2.